



Lab - Mashups and Clean Up

Modify Route Rules*

- Modify the baasRoute route rule, to allow access for /trucks:

```
<RouteRule name="baasRoute">  
  <TargetEndpoint>baasTarget</TargetEndpoint>  
  <Condition>(proxy.pathsuffix MatchesPath "/ratings") or  
    (proxy.pathsuffix MatchesPath "/trucks/*")</Condition>  
</RouteRule>
```

NOTE: only do this if you want to get the trucks from your BaaS, this is from a longer training session

Mashup the Service Callout and Target Responses

- Create another Assign Message Policy
- Mashup the ServiceCallout (ratings) and Target (trucks) responses

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<AssignMessage async="false" continueOnError="false" enabled="true"
name="Assign-Message-Mashup">
  <DisplayName>Assign Message-Mashup</DisplayName>
  <Properties/>
  <Set>
    <Payload contentType="application/json">
      {"truck":{"response.content"},
       "rating":{"ratingsResponse.content"} }</Payload>
    </Set>
    <IgnoreUnresolvedVariables>true</IgnoreUnresolvedVariables>
    <AssignTo createNew="false" transport="http" type="response"/>
  </AssignMessage>
```

Test

- Attach to Flow /trucks, Proxy Endpoint default, Segment = Response
- Test
- Verify the response contains both the ratings and trucks response mashed together

Clean Up

- There is a lot of extra metadata we don't care about in the ratings response from the Service Callout
- One method of cleansing your json of unwanted fields is to use the JavaScript policy, or you could extract out the fields you want with an extract variables policy, or a combination of both.

New Policy: **js** JavaScript ✕

There are no scripts available to configure this policy.

Policy Display Name

Policy Name

Script File

Script Name

Attach Policy ☒

Flow

Segment ☒ Request ☐ Response

Javascript for Clean Up

```
var servicecalloutcontent =  
context.getVariable('ratingsResponse.content');  
var ratingsSourceObj = JSON.parse(servicecalloutcontent);  
var cleanRatings = {};  
cleanRatings = ratingsSourceObj.entities;  
  
for (var i = 0; i < cleanRatings.length; i++) {  
    delete(cleanRatings[i].uuid);  
    delete(cleanRatings[i].type);  
    delete(cleanRatings[i].created);  
    delete(cleanRatings[i].modified);  
    delete(cleanRatings[i].metadata);  
}  
context.setVariable('ratingsResponse.content',  
JSON.stringify(cleanRatings));
```

- JS policy needs to go AFTER the Callout, but BEFORE the Mashup.

Test

- Verify the response contains the cleaned up ratings response and trucks response mashed together

See

The screenshot shows a REST client interface with a GET request to `{{HOST}}/v1/eatery/trucks/AMS1234?apikey={{clientId}}`. The response status is 200 OK. The response body is displayed in JSON format, showing details for a truck named 'AMS1234' and a rating by 'Kurt Kanaskie'.

```
1 {
2   "truck": {
3     "name": "AMS1234",
4     "displayname": "Steves Killer Burgers",
5     "color": "red",
6     "description": "TBD",
7     "cuisines": [
8       {
9         "name": "american",
10        "link": "/trucks/0d64dfff-53f0-11e6-b9c5-12cbbad10cbf/cuisines/9239a096-53f2-11e6-a496-0a65ea91a1eb"
11      }
12    ]
13  },
14  "ratings": [
15    {
16      "name": "AMS1234",
17      "comment": "Great food, great service",
18      "commenter": "Kurt Kanaskie",
19      "score": 4.9,
20      "truck": "AMS1234"
21    }
22  ]
23 }
```


THANK YOU